CHAPTER 11

MongoDB Limitations

"When starting with a new database, you should also be aware of its limitations to better use the database."

In this chapter, we will list MongoDB's limitations and the use cases where it's not a good fit.

MongoDB Space Is Too Large (Applicable for MMAPv1)

Let's start with the issue of disk space. MongoDB (with storage engine MMAPv1) space is too large; in other words, the data directory files are larger than the database's actual data.

This is because of preallocated data files. This is by design in order to prevent file system fragmentation.

The files in the data directory are named as <dbname>.0, <dbname>.1 and so on. The size of the first file as allocated by the mongod is 64MB; all subsequent file sizes increase by factor of 2, so the second file will 128MB, the third file will be 256MB, and so on until it reaches 2GB, post which all files will be 2GB in size. Though the space is allocated to the data files while creation, there might be files that are 90% empty. This unused allocated space is mostly small for larger databases.

- This option can be disabled by using the `-- noprealloc` option. However, it's not recommended to use this on a production environment, and it's supposed to be used only for testing and with small data sets where drop databases are called frequently.
- **Oplog**: If mongod is a Replica set member, then there will be a file named `oplog.rs` in the data directory. This file is present in the local database and is a preallocated capped collection. On a 64-bit installation, the allocation for this file defaults to approximately 5% of disk space.
- **Journal**: The journal files are also contained in the data directory that stores the writes on the disk before the same can be applied to the databases by MongoDB.
- MongoDB pre-allocates **3GB** of data for journaling, which is over and above the actual database size(s), making it not fit for small installations. The workaround available for this is to use `-smallflags` in your command line flags or `/etc/mongod.conf` files until you are running in an environment where you have the required disk space. But this feature makes it not fit for small installations.
- **Empty Records**: When the documents or collections are deleted, the space is never returned back to the operating system; instead, MongoDB maintains a list of these empty records, which can be reused.

  To reclaim this deleted space, either the `compact` or `repairDatabase` option can be used but be aware that both options require additional disk space to run.

S.G. Edward and N. Sabharwal, *Practical MongoDB*, DOI 10.1007/978-1-4842-0647-8_11

■ **Note** No such limitation exists with the WiredTiger storage engine. Instead, storage size reduces by 50% due to compression of data files. Also, once the collection is dropped, disk space is automatically reclaimed, which is unlike the MMAPv1 storage engine mentioned above.

Memory Issues (Applicable for Storage Engine MMAPv1)

In MongoDB, memory is managed by memory mapping the entire data set. It allows the OS to control the memory mapping and allocate the maximum amount of RAM. The result is that the performance is non-optimal and the memory usage cannot be effectively reasoned about.

1. Indexes are memory-heavy; in other words, indexes take up lot of RAM. Since these are B-tree indexes, defining many indexes can lead to faster consumption of system resources.

2. A consequence of this is that memory is allocated automatically when required. In a shared environment, it's trickier to run the database. In general, as with all database servers, it's best to run MongoDB on a dedicated server.

32-bit vs. 64-bit

MongoDB comes with two versions, 32-bit and 64-bit.

Since MongoDB uses memory mapped files, the 32-bit versions are limited to storing only about 2GB of data. If you need more data to be stored, you should use the 64-bit build.

Starting from version 3.0, commercial support for 32-bit versions is no longer provided by MongoDB. Also, the 32-bit version of MongoDB does not support the WiredTiger storage engine.

BSON Documents

This section covers the limitations of BSON documents.

- **Size limits**: As with other databases, there's a limit to what can be stored in the document. The current versions support documents up to 16MB in size. This maximum size ensures that a document cannot not use excessive RAM or excessive bandwidth while in transmission.

- **Nested depth limit**: In MongoDB, no more than 100 levels of nesting are supported for BSON documents.

- **Field names**: If you store 1,000 documents with the key "col1", the key is stored that many times in the data set. Although arbitrary documents are supported in MongoDB, in practice most of the field names are the same. Keeping short field names is considered a good practice for optimizing the usage of space.

Namespaces Limits

Be aware of the following limitations from the namespace perspective.

- **Length of a namespace**: The length of each namespace including collection and database name must be smaller than 123 bytes.
- **Namespace file size** (applicable for the MMAPv1 storage engine): A namespace file size cannot be greater than 2047MB. The default size is 16MB; however, this can be configured using the `nssize` option.
- **Number of namespaces**(applicable for the MMAPv1 storage engine): Number of namespace = (namespace file size/628). A namespace file of 16MB will support approximately 24,000 namespaces.

■ **Note** No such limitations exist for the WiredTiger storage engine.

Indexes Limit

This section covers the limitations of indexing in MongoDB.

- **Index size**: Indexed items cannot be greater than 1024 bytes.
- **Number of indexes per collection**: At the most 64 indexes are allowed per collection.
- **Index name length**: By default the index name is made up of the field names and the index directions. The index name including the namespace (which is the database and the collection name) cannot be greater than 128 bytes.

  If the default index name is becoming too long, you can explicitly specify an index name to the `ensureIndex()` helper.
- **Unique indexes in sharded collections**: Only when the full shard key is contained as a prefix of the unique index is it supported across shards; otherwise, the unique index is not supported across shards. In this case, the uniqueness is enforced across the full key and not a single field.
- **Number of indexed fields in a compound index**: This can't be more than 31 fields.

Capped Collections Limit - Maximum Number of Documents in a Capped Collection

If the max parameter is used for specifying the maximum number of documents in a capped collection, it can't be more than 232 documents. However, if no such parameter is used, there's no limit on the number of documents.

Sharding Limitations

Sharding is the mechanism of splitting data across shards. The following sections talk about the limitations that you need to be aware of when dealing with sharding.

Shard Early to Avoid Any Issues

Using the shard key, the data is split into chunks, which are then automatically distributed amongst the shards. However, if sharding is implemented late, it can cause slowdowns of the servers because the splitting and migration of chunks takes time and resources.

A simple solution is to monitor your MongoDB instance capacity using tools such as MongoDB Cloud Manager (flush time, lock percentages, queue lengths, and faults are good measures) and shard before reaching 80% of the estimated capacity.

Shard Key Can't Be Updated

The shard key can't be updated once the document is inserted in the collection because MongoDB uses shard keys to determine to which shard the document should be routed. If you want to change the shard key of a document, the suggested solution is to remove the document and reinsert the document when he change has been made.

Shard Collection Limit

The collection should be sharded before it reaches 256GB.

Select the Correct Shard Key

It's very important to choose a correct shard key because once the key is chosen it's not easy to correct it.

■ **Note** What's considered a wrong shard key depends completely on the application. Say the application is a news feed; choosing a timestamp field as a shard key would be a wrong shard key because this will end up inserting, querying, and migrating data from one shard only, and not from the complete cluster. If you need to correct the shard key, the process that is commonly used is to dump and restore the collection.

Security Limitations

Security is an important matter when it comes to databases. Let's look at MongoDB limitations from security perspective.

No Authentication by Default

Although authentication is not enabled by default, it's fully supported and can be enabled easily.

Traffic to and from MongoDB Isn't Encrypted

By default the connections to and from MongoDB are not encrypted. When running on a public network, consider encrypting the communication; otherwise it can pose a threat to your data. Communications on a public network can be encrypted using the SSL-supported build of MongoDB, which is available in the 64-bit version only.

Write and Read Limitations

The following sections cover important limitations.

Case-Sensitive Queries

By default MongoDB is case sensitive.

For example, the following two commands will return different results: `db.books.find({name: 'PracticalMongoDB'})` and `db.books.find({name: 'practicalmongodb'})`. You should ensure that you know in which case the data is stored. Although regex searches like `db.books.find({name: /practicalmongodb/i})` can be used, they aren't ideal because they are relatively slow.

Type-Sensitive Fields

Since there's no enforced schema for documents in MongoDB, it can't know you are making a mistake. You must make sure that the correct type is used for the data.

No JOIN

Joins are not supported in MongoDB. If you need to retrieve data from more than one collection, you must do more than one query. However, you can redesign the schema to keep the related data together so that the information can be retrieved in a single query.

Transactions

MongoDB only supports single document atomicity. Since a write operation can modify multiple documents, this operation is not atomic. However, you can isolate write operations that affect multiple documents using the isolation operator.

Replica Set Limitations - Number of Replica Set Members

A replica set is used to ensure data redundancy in MongoDB. One member acts as a primary member and the rest act as secondary members. Due to the way voting works with MongoDB, you must use an odd number of members.

This is because a node needs majority of votes to become primary. If you use an even number of nodes, you will end up in a tie with no primary being chosen because no one member will have the majority of vote. In this scenario, the replica set will become read only.

You can use arbiters to break such ties. They can help support failover and save on cost. To learn more about replica set functioning, please refer to Chapter 7.

MongoDB Not Applicable Range

MongoDB is not suitable for the following:

- Highly transactional systems such as accounting or banking systems. Traditional RDBMS are still more suitable for such applications, which require a large number of atomic complex matters.
- Traditional business intelligence applications, where an issue-specific BI database would generate highly optimized queries. For such applications, the data warehouse may be a more appropriate choice.
- Applications requiring complex SQL queries.
- MongoDB does not support transactional operations, so a banking system certainly cannot use it.

Summary

In this chapter, you learned about MongoDB's limitations and the use cases where it's not a good fit.

In the next chapter we will cover the How To's of MongoDB.